FORM INTERFACE AND EDITOR FOR RELATIONAL DATABASE

A Thesis Submitted
In Partial Fulfilment of the Requirements
for the Degree of
MASTER OF TECHNOLOGY

BY
VINAY MOHAN SHRIVASTAVA

TO THE

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY KANPUR

FEBRUARY, 1985

EE-1985-M-SHRI-FORM

CERTIFICATE

This is to certify that the thesis entitled 'Form Interface and Editor for Relational Database' is a report of work carried out under our supervision by Vinay Mohan Shrivastava and that it has not been submitted elsewhere for a degree.

[illegible] 25/1/85

(R. Raghuram)
Assistant Professor
Department of Elect.Engineering
I.I.T. Kanpur-208 016.

[illegible] 25/1/85

(Rajeev Sangal)
Assistant Professor
Department of Computer Science and Engineering
I.I.T.Kanpur-208 016.

ACKNOWLEDGEMENTS

I am extremely grateful to Dr. R. Sangal and Dr. R. Raghuram, my thesis supervisors, for their constant and patient help and guidance. It is difficult for me to express my feelings in words.

I would also like to acknowledge my gratitude to J.K. Marwaha for his help in implementing the system.

Thanks to all those friends who made my stay here a pleasurable and memorable experience.

I also acknowledge Mr. C.M. Abraham for his neat typing of the manuscript.

VINAY MOHAN SHRIVASTAVA

ABSTRACT

One of the widespread uses of computers is in storing and retrieving of large amounts of data. An intermediary (i.e., a programmer or a systems specialist) is necessary before a user can use the computer. This increases the response time, cost and inconvenience to the user. To address nonprogrammer community, we have implemented Form Interface and Editor system for the relational database. The system is screen-oriented and interactive. It allows a user to define the format of a form (i.e., a blank form) on the terminal screen. The form is displayed on ordinary terminal screen; graphic terminal is not required.

The system allows user to fill a form, alter the data in a form, enter the data as a tuple in a relational database and modify the format of a form. User can view a desired form on the terminal screen. If a form is large then upward and downward scrolling helps in viewing the hidden portion of the form. We have implemented paths, a facility by which the Return key steps us through the cells in a predefined path. The facility for printing, saving into a file, loading from a file and redisplaying the format of form is also available.

CONTENTS

CHAPTER 1

INTRODUCTION

1.1 MOTIVATION

One of the widespread uses of computers is in storing and retrieving of large amounts of data. As the computers provide fast access and easy manipulation of data, even the nonprogrammers have started using the computer for storing data.

A database is a collection of data stored in a computer. A Database Management System (DBMS) is the software which provides efficient storage and access to data, on one hand an and an abstract view of the data, on the other hand. It allows one or more users to modify or retrieve the data using operators in the abstract view.

If a nonprogrammer user wishes to store and manipulate data, he normally seeks the help of a programmer or a systems specialist. This introduces an intermediary between the user and the computer. It increases

(i) response time in answering the queries of a user as the time will be consumed in developing and debugging the required programs first.

(ii) cost.

(iii) Inconvenience caused due to unavailability of the specialist.

A better solution is to have a programmerless machine i.e., an easy to use interface. As it needs no programming so no intermediary will be there. Here, the interface helps in handling large amounts of data by displaying it on the screen.

Nowadays , form system is popular in handling large amounts of data as it permits one

(i) to enter only the necessary relevant information. This is because a form has fixed and precise amount of information.

(ii) to view the desired information easily. This is because a form has a fixed position where a particular information appears.

The solution is to display data through a form and store, retrieve, update the data by an editor corresponding to the form interface.

1.2 OBJECTIVES

The following are the objectives of our system.

1. To design a form interface which displays data as filled form.

2. To design an Editor (corresponding to the form interface) by which one can enter, modify or view the data of a form.
3. As for as possible, the functions of various keys present on the keyboard must remain same as normal.
4. The interface should be easy to learn.

1.3 APPROACH AND SOLUTION

The data on the secondary storage can be viewed through a suitable data model i.e., relational, network or hierarchial. Since the relational data model is simple and was available, we adopted this model for viewing the data.

1.3 ACHIEVEMENTS

Form interface and editor system based on the relational data model has been implemented. Here the user is allowed to enter, display and edit data through a form.

A form has an optional header and a sequence of pairs consisting of field name and value. The set of field names are same for all the forms of a given type. The field name indicates the type of information to be filled as the field value. A given field name always occupies the same position in the sequence of pairs. A blank form specifies the format of the form. It is like a form except instead of field value, we have the field format.

Form is displayed on the ordinary terminal screen i.e. graphic terminal is not needed. Data is entered by first positioning the cursor in the desired field of a form and then typing the characters. Thus, the system is screen-oriented. The system allows a user

(i) to define the format of a form (i.e., blank form) interactively. It involves

  (a) moving the cursor to somè desired position on the screen and typing a string of characters representing the header. Typing an arrow key ends this field.

  (b) creating all the pairs. Cration of a pair of fields involve

    1. moving the cursor at a desired position on the screen.
    2. typing a string of characters representing the field name.
    3. typing an arrow key so as to end field name.
    4. typing characters representing the field format.
    5. typing an arrow key so as to end field format.

(ii) To move the cursor

  (a) within a field

  (b) to any field of a form. This kind of movement is done by arrow and Return keys. We have implemented paths, a facility by which the Return key steps us through the fields in the predefined path.

(iii) to fill a form by moving the cursor to a desired field.

(iv) to alter data of a form.

(v) to see only part of the form in case za form does not fit on the screen. Both upward and downward scrolling have been implemented to provide easy access to the hidden parts. (which get visible if the cursor is moved to a hidden part).

(vi) to modify the format of a form.

(vii) to define four blank forms at a time.

(viii) to alter the set of keyattributes and modify their sequence of preference. Multiple attributes making a key are permitted in the system.

(ix) to enter data (of a form) as a tuple.

Brief summary lines are displayed on the screen. These display the given command and the corresponding dialogue (including the error messages in case of an error). Thus the system is interactive. It provides help at all times. The facilities for printing, saving into a file, loading from a file and redisplaying a desired blank form are also incorporated in the system.

1.4 OUTLINE OF THE THESIS

The following is the organization of the thesis.

The first chapter provides an introduction to our theis work.

Background for the work is discussed in the second chapter. In the first part, user interfaces and editors and then the file interface are discussed.

The third chapter deals with design considerations. The first part discusses the objectives in detail. Secondly, the design choices are discussed. Lastly, the various commands and their corresponding messages are given in user manual.

The fourth chapter deals with the implementation. Data structures are given in the first part and program structures thereafter. The program structures explain the flow of control and data in our system.

The fifth chapter concludes our thesis work. In the first part, the summary of the work is given. Then, the limitations and future works are discussed.

CHAPTER 2

BACKGROUND

2.1 USER INTERFACES AND EDITORS

An Editor uses power of computer for creation, addition, deletion and modification of text material such as program statements, manuscript text and numeric data. The editors allows text to be modified and corrected many orders of magnitude faster and more easily than would manual correction. No longer are editors thought of as tools only for programmers, it is now increasingly realized that the editor should be considered as the primary interface to the computer for all type of 'knowledge-workers', as they compose, organize, study and manipulate computer based information. An interactive editor is a computer program that allows a user to create and revise a target dudocuments. The term document includes the targets such as computer programs,text, equations, tables,diagrams, line art and half tone or colour photographs anything that one might find on a printed page. The document editing process is an interactive user-computer dialogue. Following are some of the functions that are available on an editor :

(i) To select what part of the target is to be viewed and manipulated.

(ii) To determine how to format this view on-line and then to display it.

(iii) To specify and execute operations that modify the target document.

(iv) To update the view appropriately.

The user interface for an editor implies the collection of tools and techniques with which the user communicates with the editor. It contains the input devices, the output devices and the interaction language of the system.

(1) INPUT DEVICES - These are used for three main purposes :

(i) To enter elements

(ii) To enter commands

(iii) To designate editable elements.

These devices as used with editors can be divided into various categories as :

(a) TEXT DEVICES - These are typically typewriter like keyboard on which a user presses and releases a key to send to the CPU or to an I/O controller a unique code for each key.

(b) BUTTON OR CHOICE DEVICES - They generate an interrupt or set a system flag, usually causing invocation of an associated application program action. Alternatively, buttons are often simulated in software by having the user choose text strings or symbols displayed on the screen.

(c) LOCATOR DEVICES - These are the x-y Analog-to-Digital transducers that position a cursor symbol on the screen by continually sampling the analog values produced by the user's movement of the device.

(d) VOICE INPUT DEVICES - These are still in research stage. It translates spoken words - both, literal text and commands to their textual equivalents.

(2) OUTPUT DEVICES - These serves to let the user view the elements being edited and the results of the editing operations. CRT terminals and hard-copy printers are the main output devices.

(3) INTERACTION LANGUAGE - It mainly has three components.

(a) LEXICAL COMPONENT - It specifies the way in which lexemes, information from the input devices or for the output devices, are combined to form the tokens used by the syntactic component.

(b) SYNTATIC COMPONENT - It specifies the input and output rules by which tokens are put together to form sentences in the grammer of the language. In terms of editors, the set of tokens might include character strings, commands and screen positions. In terms of output, the set of tokens might include character strings, lines and formatted paragraphs.

(c) SEMANTIC COMPONENT - These specify functionality. What operations are valid for each element, what information is needed for manipulation of each element. What the results of operation are and what errors may occure.

In an editor system, the command language processor accepts input from the user's input devices lexically analyzes the input stream of characters and generates a stream of tokens, syntatically analyzes the accumulated stream of tokens and upon finding a legal composition of tokens, invokes the appropriate semantic routines. Finally, the semantic routines invoke any of the user operation.

The user operations fall into several subcategories. Traveling operations allow the user to browse through document. Viewing operations allow the user to control what subset of target data is presented to the user and how it is formatted. Edit allow him to manipulate target elements.

(1) Traveling can be a simple movement i.e. position dependent traveling or it can be by pattern searching i.e. content dependent specification of location. Some editors provide the facility to travel between 2 or more files in on-line authoring i.e. interfile motion.

(2) The viewing component creates an up-to-date view on the display. Viewing operations are of 3 sorts -

(i) filtering operations, in which appropriate portions of the raw data structure of the file are selected for the viewing buffer.

(ii) operations that format these filtered data to produce an ideal view.

(iii) operations that map this ideal view to a window or page on a physical output device.

(3) The editing operation mainly involves :

(i) Creating - It involves the creation of a document. Display editors generally offer one of two kinds of input styles; type over mode or insert mode. In typeover mode, each typed character replaces the character at which the cursor is pointing.

In insert mode, each time a user types a character, the character at the cursor position and all those to its right are shifted right by one character and the typed character is inserted at the cursor position. This insertion can always be done without any commands.

(ii) DELETING - The delete command allows the user to delete the character present at the current position of the cursor.

(iii) CHANGING - The simplest change is the replacement of one letter with another. In typeover mode of a display editor, the correction is made by simply typing the new character over the erroneous one. In insert mode of a display editor, the correction is made by typing the new character and then using a delete-char command to delete the old character which has been pushed to the right by the insert of a new character.

2.2 VARIOUS EDITORS

Editors can be differentiated from the view point of target applications for which they were designed, the elements and their operations, the nature of the interface and the system configuration. These categories don't form strictly independent axes. The choice of one frequently influences the choice of another.

A text editor is one of the basic components of a text processing system, which is concerned not only with creation and maintenance but also with formulating and interactive presentation of text. Program editors operate on programs, whether represented in textual form or in another canonical form such as a parse tree. Picture or graphics editor facilitates the creation and revision of computer based graphics.

A new development in the text processing field is the document preparation system, which integerates text editing; picture editing and formating. A voice editor is a specialized interactive editor in which the target is digitally encoded voice. Some more editors systems are as follows :

(1) Line Editors : In 1960s, interactive line editors were designed that allowed the user to create and modify disk files line by line. They have problem of inability to edit a string crossing line boundaries, and inability to search for a pattern crossing line boundaries.

The following are common examples of Line Editors.

(A) IBM's CMS Editor : It is a fixed-length line oriented editor with a textual interface, designed for a time-sharing system in which terminals lack cursor motion keys and function keys. It presents the user with a one line editing buffer (the amount of the document that can be edited at a given time), although this is extended for some operations. It, also, presents the corresponding one-line viewing buffer (the amount of the document that is used to construct the display). The display is the simple mapping of one-line viewing buffer to one-line window. The following are the salient features of this editor.

(i) It has Edit and Input modes. Edit mode gives the user access to all the functional capabilities of the editor, including the capability to switch to input mode. Input mode, however, only gives the user two options : typing in text, which is simply inserted into the file at the current line pointer or pressing dual carriage returns, which returns the user to edit mode.

(ii) It provides the ability to set-up logical tab stops - tabs implemented in the editor software rather than in terminal hardware - so that tabs may be specified by typing a user - chosen logical tab character in the input stream.

(B) SOS : The editing buffer defaults to one line, although for most SOS commands a user can specify a line number or range of line numbers to expand this editing buffer. The default viewing buffer is a line.

The salient features of SOS are given as follows :

(i) The major unit of manipulation is the line.

(ii) The commands are typed in prefix notation (Verb/noun) unlike the CMS editor.

(iii) It attaches fixed, visible line numbers to each line in a file being edited.

(iv) It allows the user to create logical pages within a file for selection and organization purposes.

(v) It has a regular-expression pattern-searching facility.

(vi) The following seven different modes of operation are possible in SOS Editor.

(a) Input mode, in which SOS accepts the text being typed and inserts it into the file,

(b) Read-only mode, in which a user can travel through a file but not modify it;

(c) Edit mode, in which a user performs editing, traveling and viewing operations;

(d) Copy-file mode, in which the user can copy part or all of a file into another one;

(e) Alter mode, in which a user can perform character-by-character intraline editing without pressing carriage return to execute the command;

(f) Alter/insert mode, in which the user can insert characters such as control characters that have special meaning to the editor;

(g) Decide mode, in which the user can make case-by-case decisions for substitute commands.

(C) UNIX ed : It is a variable-length line editor. It has a single-line viewing buffer but, like SOS, ed allows the user to expand the editing buffer for a command by specifying a range of line numbers in the form starting, ending as an optional prefix to each command. The following are the salient features of ed.

(i) Like the CMS Editor, ed has two main modes edit mode and append mode.

(ii) As in the CMS Editor, lines in ed have varying internal numbers. Thus traveling is done as in the CMS Editor, with both absolute and relative specifications and ith context pattern specification as well.

(iii) It provides a facility for user-specified regular expressions in patterns defining the scopes of operations (as opposed to other editors, which use regular expressions simply for search commands).

(iv) It has the ability to reference the scope of an operation indirectly in another operand of that operation.

(2) Stream Editors : They solved the problems of Line Editors by eliminating the boundaries altogether. They act upon a document as a single conteneous chain of characters as if the entire document were a single indefinitely long character string, rather than act upon fixed length or variable length lines. TECO, described below, is the popular editor of this category.

(A) TECO : TECO, the Text Editor and Corrector is an interpreter for a string processing language. The editing buffer is the amount of the file in memory. The viewing buffer on the document defaults to the null viewing buffer. The document is displayed only on the explicit command; the user can specify a viewing buffer of any size. The following are the salient features of TECO.

(i) Fundamental commands such as insert, delete and context search are available.

(ii) It supports commands for conditional execution to aid in creating more complex commands. Q-registers are available for holding any numeric or string value.

(iii) Although TECO is character oriented, special commands allow the user to edit a document in terms of a line model.

(3) Display or Cursor Editors : These allow a cursor to be moved anywhere on the text displayed on the multiline screen, i.e., a user is able to move the cursor to point to the text, he wishes to manipulate. Text is conceived of as a quarter - plane extending indefinitely in width and length, with the topmost, leftmost character the origin of the file. The user travels through this plane by using cursor keys and changes characters by overtyping. Text is input on the screen at the position of the cursor. The editing and viewing buffers can be moved left and right and multiple windows support easy interfile editing. These make use of pick and delete buffers.

The salient features of these editors are given below:

(i) It is modeless, since all typing on the screen is considered text, commands must be entered either through function keys, control characters and escape sequences, or by moving the cursor to and typing in a special command line at the bottom of the screen.

(ii) The command syntax is single operand postfix.

(iii) It has a marking facility which allows the user to select with the cursor two arbitrary points in the text to define a scope not easily specified with the element modifiers.

(4) Syntax Directed Editors : These editors have the knowledge of the syntax of the thing (e.g., a pascal program) being edited. These are structure editors that ensure that the structure always is constrained to preserve syntactical integrity.

We have implemented a form interface to the database. So, the kind of editor present in our system is the form editor. It is an interactive editor and allows the user to enter, display and modify the data of a form.

The data of a filled form is stored in the database by the 'ACCESS METHOD INTERFACE'. The interface provides the functions to access the data in the database efficiently and uniformly.

2.3 ACCESS METHOD INTERFACE

This interface is based on the Relational Data Model. AMI is the interface between the user and the file structures providing the structural transparence, i.e., it gives the user an independent view of the data, free from the underlying file structure. This is a set of standard functions provided to the user to gain access to the data stored in the database. Following are the various functions.

1. OPENR (Chno, Fname)

This function opens the file name specified in Fname and assigns a channel no. to it, which is used in other routines of AMI. For any legal operation on the database, the relation has to be opened. This sets up all the control structures for the corresponding file.

2. GETNEXTRECORD (Chno, Adrs)

This function retrieves the next record from the file with channel no. chno and put it at the location starting with Adrs.

3. PUTNEXTRECORD (Chno, Adrs)

This function writes at the next position in the file with channel No. Chno taking the record from location starting with Adrs.

4. GETRECORD (Chno, Adrs, Keyval)

This function retrieve the record with key-value keyval in the database and if found, puts it at the place pointed to by Adrs.

5. INSERTRECORD (Chno, Adrs)

This function inserts a record in a file with channel no. Chno taking the record stored starting from Adrs. It calculates the Keyvalue from the record itself and doesn't allow duplicate records.

6. DELETERECORD (Chno, keyval)

This function deletes a record with key-value keyval from a file with channel no. Chno.

7. UPDATERECORD (Chno, Adrs)

This function updates a record (only for non key fields) of a file with channel no. Chno byttaking the record from location starting with Adrs.

8. MODIFYRECORD (Chno, Adrs)

This function modifies a record (modifies key fields also) of a file with channel no. Chno by taking the record from location starting with Adrs.

9. CLOSER (Chno)

This function closes the file with channel no. Chno for any further operation on it.

10. CREATER (Fname, Flag, Atrcount, Adrs1, Adrs2)

This function creates a file name specified in Fname. The value of Flag 0 or 1 determines the HASHED or ISAM file structure respectively. Total no. of attributes are specified in Atrcount. Adrs1 and Adrs2 specify the no. of non key attributes and key attributes respectively.

As the various AMI functions are presently available for CP/M CROMEMCO system and expected to be available for DEC-10 system near MAY, 1985. So, in the present interface, we simulated the above functionality routines of our design. Here, some portion of main memory acts as secondary storage.

CHAPTER 3

DESIGN CONSIDERATIONS

A form has an optional header and a sequence of pairs consisting of field name and value. The set of field names are the same for all the forms of a given type. The field name (or name field) indicates the type of information to be filled as the field value (or value field). A given field name always occupies the same position in the sequence of pairs.

A blank form specifies the format of the form. It is like a form except instead of field value we have the field format.

1. ROLL NO : -----	1. ROLL NO: 84104
---	351
(Fig.(i)	(Fig.(ii)
(For a blank form)	(For a filled form)

Fig.(i) shows a pair of fields of a blank form.
'1. ROLL NO:' is the field name.
'-----' is the field format (or field value). The display of the data (84104351) to be entered into this field, will follow these lines as shown in Fig. (ii) (i.e., a character will be displayed at the position of '-' only). The lines representing a field format of a blank form may be of any

character e.g. '-', '.', '*' etc. We have chosen '-'. Entering the data in the field value of a form is known as filling a form.

In a Relational Database system, the field names correspond to attribute names and the field values in a form correspond to a tuple.

3.1 OBJECTIVES

(a) The form-interface and editor must allow us to

(i) Define the format of a form interactively

(ii) Modify the format of a form

(iii) Fill a form

(iv) Alter the data of a form

(v) Enter the data as a tuple in the relational database.

(b) While designing commands for above actions, the function of various keys present on the keyboard must remain as normal as possible.

(c) The interface should be easy to learn.

3.2 DESIGN CHOICES

To perform the above tasks by the form-interface and editor system, the following design choices are made.

1. To define a form means creating a blank form. A blank form creation involves specifying -

(i) The starting point of the form and its header.

(ii) All the pairs of a form. Specifying a pair includes to specify field name, field format and their starting positions.

As a field format follows the field name, so there is no need to specify the starting position of field format. One way to define the format of a form is to specify all the above information before the display of form. This can be done by typing.

(i) Name of header and its starting point on the screen.

(ii) The information for all the pairs of fields. A pair of field can be specified by typing.

(a) Field name and its starting point,

and (b) Field value in terms of

- no. of dashes in a line
- no. of spaces for each discontinuity (---b̸b̸b̸---)
  Here blanks represents discontinuity in the line represented by dashes.
- position of dashes and spaces.

This method is cumbersome in specifying field format. As, the form will be displayed after entering all the specifications, so one can't visualize it during creation. Hence modification of field name or field format is difficult.

We adopted an interactive kind of form creation. This method is screen oriented, as the user is free to move the cursor and free to type anywhere on the screen. It involves

(i) moving the cursor to some desired position on the screen and typing a string of characters representing the header. Typing an arrow key ends this field.

ii) Creating all the pairs. Creation of a pair of fields involve -

- (a) moving the cursor at a desired position on the screen,
- (b) typing a string of characters representing the fieldname
- (c) typing an arrow key so as to end fieldname
- (d) typing dashes, spaces and return in order to create a line, discontinuity and starting a new line respectively,
- (e) typing an arrow key so as to end field format.

This method eliminates most of the limitations of the previous method. The modification of field name or field format is easy, as the created form always remains displayed on the screen.

The data structure of a form should store all the information necessary for its display. The following record type can store a form.

FORMHEADER	- stores header of the form
ROWH,COLH	- starting point of the header
FORM-content	- an array of record having the following

NAME	- stores field name
ROW,COL	- stores starting point of field name
VAL	- stores field value

The key attributes of the system are those attributes whose values determine a tuple in the database uniquely. We decided to represent a key attribute by entering

'*' character at the starting of the corresponding field name. Any attribute can be deleted from or included in the set of key attributes by deleting or inserting '*' at the starting of its corresponding field name.

2. Filling of form involves the entering of data into alue fields. For this firstly, one has to put the cursor in the alue field. So cursor should be allowed to move to any of the value fields. We have used the following arrow keys for this type of movement.

Moveup '↑'	- For moving the cursor to the previous 'alue field.
Movedown '↓'	- For moving the cursor to the next alue-field.

If cursor is present in the first or last value field of the form being displayed, then these arrow keys should scroll the form in the corresponding directions (i.e., '↑' - upward and '↓' - downward) and display the hidden data (of that direction) roughly in the middle of the screen. So the hidden data may be viewed along with some of the surrounding data.

To modify the contents of a value field, characters need to be deleted or inserted. Insertion can be done by putting the cursor at the desired position and typing the desired character. Deletion can be done by putting the cursor at the position next to the character to be deleted and giving delete command by 'delete' key. So the ability to move the cursor within the cell is essential. Cursor can be moved to the previous character by 'backspace' key and to the next character by ^A.

To enter data into alue fields of a form, the cursor has to be moved through each of these.

This involves a lot of hand-movement as the arrow-keys are are located far away from the other keys. Moreover, one would have to decide the proper arrow key and its sequence for a desired movement. We used return key to avoid the above difficulties. Return key positions the cursor in the next available value field of the same form. If it is the last value field of the form, the cursor is put in the first value field of the next form.

Sometimes, need may arise to enter or modify the data of some particular fields for a number of forms. So, there is no need to move the cursor through all the value fields of the form. In such a case return should place the cursor in the same value field of the next form.

If one wants to enter or modify the data of some selected value fields of a number of forms in a desired sequence, he must first specify the sequence of value field numbers. The return key now outs the cursor in the value field whose value field no. is next in the sequence in the same form. If it is the last value field of sequence, the cursor is put in the next form in the first value field of the sequence.

Such movements are known as following a predefined path. Return key steps us through the cells in the paths.

A command is needed to define the paths.

3. There should be a command for altering the format of the form. It involves either name field (it includes the modification of keys also) or value field or both the fields modification. So, the cursor should be allowed to move to any fields. This is done by the following arrow keys.

Moveright - It places the cursor in the next field.

The next field may be value field of same form- content or name field of next form content.

Moveleft - It places the cursor in the previous field. The previous field may be value field of previous form-content or Name field of same form-content.

4. Additional Commands needed are as follows :

(i) There should be a command to display a desired tuple.

(ii) In case there are more than one type of forms, there should be a command to display a desired type of form (i.e., desired blank form.)

(iii) There should be a command to delete a tuple from the relation.

(iv) There should be a command to delete the format of a form or a relation (i.e., delete the data of all the forms.)

There might be some additional commands to facilitate some of the operations. Such commands might be -

(i) To redisplay the format of form i.e. blank form .

(ii) To save the data in the current session in order to preserve it. (As we are simulating the secondary storage in the main memory.)

(iii) To load a session from secondary storage.

(iv) To print a form.

(v) To ask for help and quit the session.

5. Brief summary lines should be displayed all the time. A summary line is needed to display the given command, and the

corresponding dialogue (including the error message in case of an error.). Another line is needed to display the overflow string from a value field. It is natural to display the overflow string from a value field below the last row of form. Moreover, all the summary lines should be displayed together. Therefore, these lines are displayed on the bottom of the screen.

3.3 USER-MANUAL

Various commands available in the system are listed below.

. On-line Commands

- Help commands.
  - (i) ?
  - (ii) ^U - Help for update commands.
  - (iii) ^V - Help for various slash commands.
- Inserting text.
- Moving about on the form.
- Going into slash mode.

The various slash commands are as follows. These follow the slash ('/') character.

(i) C - create command

(ii) U - update

(iii) R - redisplay a blank form

(iv) M - moving to another type of blank form

(v) W - way or path

(vi) F - find

(vii) D - delete

(viii) K - kill

(ix)a. S - saving a form in a default filename

b. ^S - saving a form in a desired filename

(x) a. L - Loading a form from a default filename

b. ^L - Loading a form from a desired filename

(xi) P - print

(xii) Q - quit

(xiii) O - open

(xiv) X - close

3.3 USER MANUAL

3.3.1 ON-Line-Commands

When the user runs the system by typing EX MM.NEW/PAS, A,INP , the 'TITLE BOX' (indicating the name of the system) appears on the screen. The system waits for input. Help is available on typing '?'. If one wants to know more details, he should type

^U for update commands,

^P for path description and

^V for various slash commands.

The information is displayed and the system waits for input. One can give slash 'C' to create a form (as described later). The cursor is positioned at the first value field. The value field can be filled by typing alphanumeric characters. The user is allowed to type the following control characters to alter the filled data or to move within a field.

1. CONTROL A (^A) : It moves the cursor to the next available character.
2. BACKSPACE (^H or Backspace key) : It moves the cursor to the previous character.
3. DELETE ('Delete' key) . It deletes the character preceeding to the current position of the cursor.
4. CONTROL X (^X) : It positions the cursor in the last character of the current field.
5. CONTROL R (^R) : It replaces the current field by Null data. Cursor is put at the starting position.

If the cursor is present in the boundary (first or last) row of the part of the form being displayed, typing arrow keys will scroll the form in the corresponding direction and display the next hidden form-content roughly in the middle of the screen.

The scrolling provides to view different parts of a form.

3.3.2 Slash Commands

Wherever the user has to enter an integer number he should give escape to end it. The commands are given as follows.

1. CREATE Command : It allows to create a blank form (representing the format of the form), interactively. The command is given by '/C'. The system responds with 'Enter Relation No.'. As soon as one enters form no., the cursor is put at the top left corner of the screen. One can start creating form by entering 'form header' field. He should press movedown or moveright arrow keys to end a field. Thereafter, enter the first name field. If it is a keyattribute, type '*' at the beginning. Now, the system accepts the value field of same FORM-CONTENT. Similar!, the user may enter W-maxcols entries in the same form. He may end the creation by typing ' E'.

2. UPDATE : It is given by '/U'. It is used to modify the format of a form. It has a power to alter the set of keys and their order of priority.

Name fields and value fields can be updated by using CONTROL commands. Moveleft and moveright keys move the cursor to the adjacent field. The user ends updating by typing ^E.

3. REDISPLAY : It is given by '/R'. It displays the blank form. The cursor is positioned at the first value field. One can fill a form by typing alphanumeric characters. They will be echoed back on the dash line only. If one desires to fill more characters than the available dashes, the overflow characters get displayed on the extension-line. By movedown or moveup keys, the user may put the cursor to the adjacent value field. He can enter a form by ^E. Thereafter, the blank form appears on the screen, again. Similarly, a user can fill W-maxrows forms in a relation. If he wants not to fill more form, he may give ^F.

4. MOVE : This command is given by '/M'. The system responds with 'Enter relation no.'. The user specify the relation no. Thereafter, the blank form is displayed. The cursor is positioned at the first value field.

5. WAY or PATH : This command is given by '/W'. It allows one to use RETURN key to follow a path. The system responds with the message, 'Enter desired path-code'. The user may enter 2,3 or 4 according to a desired path movement.

(i) PATH CODE 2 : Path 2 allows to move in the next available value field. It is useful to fill a blank form when the cursor is in the last FORM-CONTENT, typing the Return will enter the form. The new blank form appears, so as the user can fill the next form. It continues till he gives ^F. The path 2 is the default path of the system.

(ii) PATH CODE 3 : Path 3 helps to view the data in a relational database. It allows to move the cursor in the same attribute present in the next tuple. If it is the last tuple, the system responds with 'No other tuple is present in the Database'. Simultaneously, a blank form appears on the screen. Moreover, the path-code is automatically switched to 2.

(iii) PATH-CODE 4 : It allows to move the cursor according to a selected attribute sequence. The system responds by the message 'Define Path'. The user should enter the attribute no's in the desired sequence, i.e., if he wants to move along a path consisting of fifth, second, fourth and seventh attribute. He should enter the no's like 5b̸2b̸4b̸7b̸b̸. Thereafter, the first form is displayed. The cursor is positioned at fifth attribute value field. Typing the return key will place the cursor at second attribute. Similarly, one can finally come at seventh attribute on giving return again, it will display the next form. The cursor is positioned at fifth attribute. The above sequence repeats until the user give return from the last tuple's seventh attribute. The blank form appears on the screen and cursor is positioned at first attribute. The path code automatically becomes 2.

(iv) PATH-CODE 1 : The user is not allowed to give it. It is used internally in creation or updating a form.

6. FIND : This command is given by '/F'. It is used to retrieve a specified tuple. The system responds with,

'Give the keyattributes in the order ..., ..., ...'

one should enter the keyattribute values in the same order as prescribed by the message.

In case the tuple is not present in the database, the system indicates it by, 'Tuple is not present'. Otherwise, the corresponding form is displayed on the screen. The cursor is put at the first value field.

7. DELETE : This command is given by '/D'. It allows to delete a particular tuple. The system responds with the message, 'Give the keyattributes in the order ..., ..., ... '.

The user should enter the set of keyattribute values in the order indicated by the message. Thereupon, he should enter total no. of tuples in case he wants to delete more tuples.

The message appears 'No. of tuples deleted ...'

8. KILL : This command is given by '/K'. It destroys the data and the format of a form. The system inquires 'Enter relation no.' As soon as, he enters relation no, the following message appears,

'Relation is destroyed'.

9. <u>SAVE :</u> There are two commands for saving the data of the current session along with the definition (or format) of a form.

(a) /S : This command is given by typing /S. System responds with the following sequence of queries.

(i) 'File' : Here user enters the name of the file into which he wants to save the data. This file is known as data file. File name may have an extension (of maximum 3 characters). In case no extension is given by the user, a default extension 'DBS' is assumed by the system. Definition (or format) of the form (to be specified in the following queries) is stored in a file whose name is same as that of data file but extension is 'FRM'. This file is known as format file. e.g., If data file is specified as WXY.Z, then data will be stored in WXY.Z and the definition of the form will be stored in WXY.FRM. If data file is specified as RAIL, then data will be stored in the file RAIL.DBS and definition of the form in the file RAIL.FRM.

(ii) 'Enter relation no.'

(iii) 'Enter format no.'

Relation no. and format no. specify the form whose definition is to be stored.

After answering the above queries, the message 'saving started' appears on the screen. When savings is completed, the message 'saving completed' is displayed.

(b) /^S : This command is given by typing /^S. This command is also used to save the data of current session along with the definition of a form. But here the definition of a form is stored in a file (called format file) specified by the user. Sequence of queries is given as follows :

(i) 'Enter data file name' : User enters the file name into which he wants to save the data. As in the previous case, here also if no extension is given, a default extension 'DBS' is assumed by the system.

(ii) 'Enter format file name' : Here user enters the file name into which he wants to save the format or definition of the table (to be specified in the following queries)

e.g., if data file is specified as WXY.Z and format file as AB.C, then data and format will be stored in these files respectively. If data file is specified as WXY and format file as AB, then data will be stored in the file WXY.DBS and the format will be stored in the file AB itself.

(iii) 'Enter relation no.'

(iv) 'Enter format no.'

After this the message 'saving started' appears on the screen and when saving is completed the message 'saving completed' appears on the screen.

10. '<u>LOAD :</u> This command is used to bring the data back into the simulated secondary storage and the definition of the form in the main memory from the files saved as above. Like 'Save' here also we have two load commands.

(a) '/L' : This command is given by typing '/L'. This is used to load the data (the definition of the form from the files saved by '/S' command. Following is the sequence of queries.

(i) 'File:' : User enters the data filename only. If the file name has the extension other than 'DBS', then he should specify the extension also.

When loading is completed, the message 'loading over' appears on the screen.

(b) '/^L' : This command is given by typing '/^L'. This is used to load the data and the definition of the form from the files saved by '/^S' command. Following is the sequence of queries.

(i) 'Enter data file name' : As described above, the user should enter the name of data file.

(ii) 'Enter format file name' : Here he enters the name of the format file. If file name has extension other than 'FRM', then he should specify this extension also.

The message 'loading over' appears on the screen after loading is over.

11. PRINT : One can use print command to take hard-copy of a form. It is given by '/P'. The system responds with the following sequence of queries by taking an input, if necessary. The various messages and their meanings are as follows :

(i) 'File' : means enter the file name.

(ii) 'Enter relation no' means enter an integer no. corresponding to desired relation, e.g., say 1,2, 3etc.

(iii) 'Enter starting Row, Col' - enter the desired starting point of the form in the file. e.g., if one wants to start form header field from 2nd row and 25th column, he should enter 2$25$.

(iv) 'Enter row separation' - cnter the additional separation required between the various name fields during print, e.g. if one wants to keep i+ same, he should enter as.

(v) 'Enter no. of copies' - enter no. of times a form should be printed. e.g., if one wants to have 3 copies, he should enter 3$.

(vi) 'Enter no. of tuples to be printed' - enter an integer no. say 3,4, etc.

(vii) 'PRINTING over'.

12. QUITS : This command is given by '/Q'. To end execution of the program, one may give it.

13. OPEN : The user has to open a relation for doing any legal operation on the relation. This command is given by '/O'.

14. CLOSE : This command disables us to view or change a relation. It is given by '/X'.

3.3.3 Error Reporting

(i) ERROR MESSAGES and Their Meaning :

If one commits an error in giving a command or pressing a relevant key, the system responds with an error message to indicate the error. It rings the bell also to draw the user attention. The following are the various error messages. The system responds with the error message

1. 'Can not move further up', in case the cursor is placed at the form header field of the form (i.e., at the first line of the form) and the user gives moveup.

2. 'Can not move further down', in case the cursor is placed at the last name or value field and the user gives movedown.

3. 'Can not move further left', in case the cursor is present in the first name field and the user gives moveleft.

4. 'Can not move further right', in case the cursor is in the last field and the user gives move right.

5. 'Character string can't be more than ...', in case user tries to fill the value field with the characters more than the specified. He should stop the further entry of characters in this field.

6. 'Tuple with the same keyattribute is already present', in case a user tries to insert a tuple, with keyattribute values,same to the tuple already existed.

7. 'No other form is present', in case the cursor is placed at the last value field of last form and the user gives return.

8. 'No other attribute can be made as key', in case the keyattributes are already four and one tries to make an addional keyattribute.

9. 'Attribute is already a key', in case a user tries to make a keyattribute which is already a key.

10. 'Press only moveup or movedown keys for movement', in case one gives moveleft or moveright without altering a field or without giving a control command or return.

11. 'The file is not present in the directory', in case one gives the name of the file which is absent in the directory.

(ii) Errors causing Monitor Intervention :

There are some errors which causes the monitor intervention.

1. If the user tries to load an arbitrary file in his directory (which has not been saved by the form interface),

the monitor intervention occurs with the message 'scalar out of range'.

2. The present implementation simulates a secondary storage in the main memory. If somehow, the main memory requirement crosses the maximum limit, the monitor intervention occurs with the message 'stack overruns heap, retry with more core'.

CHAPTER 4

IMPLEMENTATION

4.1 DATA STRUCTURES

1. FORMRECORD:

The following information is associated with a form :

i) Header

ii) Screen position of Header

iii) Field Name and Field value of each of the pairs in it

iv) Screen position of Field Name of each pair.

Screen position is represented by a pair of row and column number of TTY screen. As each of Header, Field Name and Field Value can be considered as a string of characters, it can be stored in a data structure of type STRING , where STRING is a packed array of characters.

```
STRING = packed array [TTYCOLS] of char;
        and TTYCOLS is TTYCOLS = 0 ... 100;
```

Now, to store a form in main memory a data structure called FORMRECORDis used. It has four fields. First field FORMHEADER stores the Header of the form and hence is of type STRING. The next two store its screen position. As a form may have a number of pairs (of Field Name and Field Value), the fourth field FORM-CONTENT is an array of type ENTRY. Each element of the array stores Field Name, Field Value and the screen position of Field Name.

```
      FORMRECORD = record
                     FORMHEADER: STRING ;
                     ROWH,COLH:  Integer;
                     FORM-CONTENT: array [1..W-Maxcols] of ENTRY
                   end;
                     W-Maxcols is the maximum no. of pairs that
                     a form can have.
      ENTRY        = record
                     NAME,VAL: STRING;
                     ROW,COL : Integer
                   end;
```

2. <u>VALREC :</u>

As the system uses the relational data model, it is necessary to enter and retrieve the data at the tuple level. A tuple consists of various attribute values. An attribute value is a string of characters and so can be stored in the data structure of type STRING. To store a tuple temporarily in the main memory VALREC data structure is used. It is an array of type STRING. Here the attributes of the tuple are the elements of the array in order.

```
      VALREC = array[1...W-Maxcols] of STRING;
```

The variables of type VALREC (i.e. VALRECORD, VALRECORD1, VALRECORD2) are used to transfer the Field name and Field Values (each) as a tuple from the secondary storage to main memory and vice-versa.

3. KEY-REL and KEY-COL :

The present implementation allows us to have multiple keyattributes. In this case the searching and sorting will be done according to the sequence of key attributes. Sequence of key attributes determines the order of priority given to each of these during searching and sorting.e.g., Suppose there are 3 key attributes A,B and C. So if their sequence is B,C,A, then the comparison between two tuples will be made first on the values of B, then on the values of C and lastly on the values of A.

The above shows that there is a need to store the sequence of the key attributes. More memory is required to store the key attribute name than that required to store the number of each key attribute (each attribute of the relation and hence the corresponding pair of this form is, associated with a unique number). Moreover, memory requirement is reduced further if the numbers are stored in the form of characters. An array key-rel of type character can be used to store the number of each attribute present in the sequence. Position of this number in the array determine the order of priority given to the corresponding key attribute.

```
Key-Rel = packed array [1 ... 7] of char;
Key-col : array [1 ... Maxrels] of Key-Rel;
```

Maxrels stores the maximum number of relations that one can have in secondary storage.

Considering the above example, if A,B and C be the key attributes of a relation No.1, the numbers associated with these are, say, 3,6 and 8 respectively. Then a sequence B,C,A will be stored as

```
Key-Col [1][1] = CHR (6)
Key-Col [1][2] = CHR (8)
Key-Col [1][3] = CHR (3)
```

As first element of the array is CHR(6), so while comparing 2 tuples first preference will be given to the corresponding key attribute i.e. 'B' and so on.

4. PATH-ATT :

In case of path 4 (ref. Chapter 3) user has to specify the sequence in which he wants to move the cursor through the desired attributes. So it is required to store these desired attributes and the corresponding sequence. An array Path-Att of integers can be used to store the number of each of the desired attributes. Positions of these numbers in the array will determine the sequence.

```
Path-Att: array[1 ... W ..Maxcols] of integer;
```

For example if the user wants to move the cursor in the sequence: 2nd, 4th and 3rd attribute, then these numbers (i.e., 2,4 and 3) will be stored in Path-Att as follows

Path-Att [1] = 2, Path-Att[2] = 4, Path-Att[3] = 3 .

5. <u>ROW-KEYS and KEYS :</u>

A relation has a number of tuples, so finding a particular tuple may require a large no. of accesses to the secondary storage. One way to reduce the no. of accesses is to store the relation in an ISAM file and build an Index. As we are simulating secondary storage in main memory, we can store only a few tuples. So an Index can be simulated by an array Row-Keys of type keys, which stores the key attributes of each tuple of the relation. (As shown below Keys stores all the key attributes of a tuple.)

```
Row-Keys = array[1 .. W-Maxrows] of Keys;
Keys     = array[1 .. Max-Keyatt] of STRING
```

W-Maxrows is the maximum number of tuples that one can have in a relation and Max-Keyatt is the maximum number of Key-attributes in a relation.

Further Keyattributes of the tuples, in the Row-Keys can be stored in the same order as the tuples in the simulated secondary storage. Therefore, the position of Key attributes in Row-Keys determines the position of the corresponding tuple in the secondary storage.

6. W-SHEET :

A relational data model can be thought of as a two dimensional structure in which one dimension (say horizontal) specifies a tuple (a row) while the other (say vertical) specifies an attribute. So an obvious choice for storing a relation is a two dimensional array. One dimension, called row, specifying a tuple and other, called column, an attribute. Both dimensions together specify the value of an attribute for a particular tuple. As the value of an attribute is a string of characters, it can be stored in a variable of type STRING. Thus the above two dimensional array can be of type STRING. To have more than one relation, one more dimension is needed. So the final choice for the data structure storing the relations is

```
W-Sheet:array[ 1...Maxrels,-2...W-Maxrows,1...W-Maxcols]
                                          of^STRING
```

Instead of STRING, here we have used^STRING to avoid unnecessary allocation of memory.

The array W-sheet is used to store both, the data of relation and the format of the form. For a relation, data is stored in rows from row no. = 1 to row no. = W-Maxrows.

As discussed earlier, each pair in the form corresponds to an attribute of the relation, the no. of pairs in the form is same as the no. of attributes in the relation. So Field

Names of all the pairs in the form can be stored as a tuple in the W-sheet. Rowno. = -1 is reversed for Field Names. Similarly Field Formats (i.e., Field Values of a blank form) of all the pairs in the form can be stored as a tuple in rowno. = -2.

For example, let a pair of a form is

```
2.Roll   No: --------
b̸b̸b̸b̸b̸b̸b̸b̸      -----
```

Fig. (i) a pair of a form

Here field name is '2. Roll No' and rest is field format Let this pair corresponds to second attribute of the relation 'V' and the no. 1.Screen position of the field name is 2,3, i.e., field name starts from the second row and 3rd column on the tty screen.

Then the Name Field will be stored as

W-sheet [1,-1,2]^.STRING = 2.Roll No:

Since a screen position is associated with the Field Name of each pair, these (i.e., screen positions of Field Names of all the pairs) can be considered as a tuple and can be stored in some particular row in W-sheet. As a screen position consists of two numbers only, reserving a full row for screen positions will be quite expensive.

So instead of having a full row for screen positions, a screen position associated with a pair is stored along with its Field Name in the same variable as shown below.

As the variable of type STRING is an array [0 ... 100] of char, screen position can be stored as characters in last two elements of this array*. So we don't need any extra storage for screen position. Of course this is achieved at the cost of reduced maximum length (reduced by 2 characters) of Field Name that one can have. Now the element W-sheet [1,-1,2]^.STRING will look like

1	2	3	4	5	6	7	8	9	10 ..		99	100
CHR (2)	.	R	O	L	L	b̸	N	O	. . .	nul	CHR (2)	CHR (3)

Field Name
'2.Rollb̸No:

Screen position
2,3

Field Formats of all the pairs of the form are stored as a tuple in row no. = -2 of the W-sheet. As shown in Fig.(i), field format consists of dashes, spaces and returns. One way of storing the field format is to store all the dashes, spaces and returns.

*Last two elements are chosen to have easy identification and non-interference of screen position with its corresponding Field Name.

e.g., field format of Fig. (i) can be stored in W-sheet [1,-2,2]^,STRING as

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
-	-	-	-	-	-	-	-	CHR (13)	CHR (10)	b̸	b̸	b̸	b̸	b̸	b̸	b̸	b̸	-	-	-

22	23	24	25
-	-	CHR (13)	CHR (10)

Here in all we need 27 memory locations to store the field format. If, instead of storing all the dashes, we store only the number of dashes, in a line as a character, then memory requirements will get reduced. As now instead of 8 locations to store 8 dahses in the first line of the field format, we need only one location to store CHR (8) (as the no. of dashes in first line is 8). The maximum possible length of a line on tty screen is 80 (as there are 80 columns as tty screen), so it can be represented by a single chara-cter, as (the maximum number which can be represented by a character is 127.) Similarly instead of storing both CHR(13) and CHR(10) for return, we can store only CHR(13). But now a line consisting of 13 dashes will conflict with a return, as both will appear as CHR (13) in an element of W-sheet. To avoid this number of dashes a line can be stored after adding 13 to it. So a 8-dashes in line will be stored as CHR(13+8) i.e. CHR(21). ASCII code of space is 32. Now a

19-dashes line will conflict with a space as both are stored as CHR (32) (19+13).

Since after addition of 13 to the no. of dashes in a line, the resultant no. will always be > 13. So a space can be stored as a character whose ASCII code is $\not>$ 13. We have chosen to store space as CHR (12).

So finally the field format of Fig. (i) will be stored in W-sheet [1,-2,-2]^ String as

CHR (21)	CHR (13)	CHR (12)	CHR (12)	CHR (12)	CHR (12)	CHR (12)	CHR (12)	CHR (12)	CHR (12)	CHR (18)	CHR (12)

With the above scheme, size of the array storing a field format can be much smaller than the size of array storing the value of an attribute or a Field Name. But in the present implementation, for simplicity, we have chosen both the arrays of same size (i.e., size of array STRING).

The functional block diagram illustrates the various controls and corresponding flow of data in the system. The entered string of characters is recognized by the COMMAND LANGUAGE PROCESSOR and further action is taken depending on the type of input string. It can be a data or a command. The data is entered at the tuple level through the editing buffer invoked by entry and editing component. The data is simultaneously displayed on the screen by 'Writechar-form' (Ref. Fig. (ii)) procedure of DISPLAY component.

If the entered character or string is a command, the COMMAND LANGUAGE PROCESSOR invokes the following appropriate component for its processing.

(i) The search component enables to find a particular tuple from the database and display it on the screen. It generates viewing buffer after processing through editing and viewing filter. Then, the contents of viewing buffer are displayed by Display component.

(ii) The Travelling component enables to move the cursor on a form. In case of scrolling, it displays the hidden part of the viewing buffer by implicitely invoking the Display component. While following a movement in PATH3 or PATH4, it generates viewing buffer in the case of a new tuple to be displayed.

It is desired that there should be no separate command to go into edit-mode (editing commands are given like data) and the editing of a form should be visual. So, editing buffer and viewing buffer should have the same data.

4.2 PROGRAM STRUCTURES

(A) The Travelling Component is used to move the cursor at the desired field of a form. It consists of subroutines named as FORMUP, FORMDOWN, FORMRIGHT, FORMLEFT and PATH-SELECTION.

These subroutines use the PUTCURSOR and ENTRY-LENGTH for placing the cursor and finding the screen position of VAL (field value) of a FORM-CONTENT in FORMRECORD data structure. The various subroutines are given as follows :

1. PUTCURSOR (Row,Col: integer);

   Function: It positions the cursor at the point defined by integers Row and Col on the screen.

   In the following part of this chapter, we will term it as cursor position.

   Global variables accessed : C-row.

   Global variables modified : C-row, C-col.

2. ENTRY-LENGTH (S: String);

   Function : It finds the screen position of field value for a given 'S' representing the field name. The subroutine uses the screen position of field name and assign the calculated screen position of field values to V-row and V-col global variables.

   In the following part of this chapter, we will term it screen position of field value.

   Global variables accessed : FORMS.FORM-CONTENT [contentindx], Row and Col.

   Global variables modified : V-row, V-col.

3. FORMRIGHT;

Function : If the cursor is assumed to be present in field name then the subroutine places the cursor in the field value. If cursor is already in value field then FORMRIGHT positions the cursor in name field of next pair.

Global variables accessed : FORM-CONTENT [CONTENTINDX], C-row, C-col, CONTENTINDX, Page, FORM-CONTENT [CONTENTINDXH], Row C-Col.

Global variables modified : C-row, C-col, CONTENTINDX, V-row, V-Col.

4. FORMLEFT;

Function : If the cursor is present in field value, the subroutine places the cursor in the field name. If cursor is already in the field name then FORMLEFT positions the cursor in the field value of previous pair.

Global variables accessed : FORM-CONTENT[CONTENTINDX], C-row, C-col, CONTENTINDX, Page, FORM-CONTENT [CONTENTINDX-1], Row and Col.

Global variables modified : C-row, C-col, CONTENTINDX, V-row, V-col.

5. FORMUP;

Function: It positions the cursor at the previous field value (Here previous signifies the pair which is just previous to the pair pointed by the cursor). It involves scrolling when the previous field value is hidden.

Global variables accessed : FORM-CONTENT[CONTENTINDX-1], CONTENTINDX, page, minindex , W-sheet.

Global variables modified : C-row, C-col, CONTENTINDX, V-row, V-col, page, miniindex, maxindex.

6. FORMDOWN;

Function: It positions the cursor at the next field value (Here next means the pair which is just next to the pair pointed by the cursor). It involves scrolling when the next field value is hidden. The scrolling at the time of creation of a form is done by storing the existing form.

Global variables accessed : FORM-CONTENT[CONTENTINDX-1], CONTENTINDX, page, maxatt, maxindex, C-row, W-sheet, Val-create, form-create, Valrecord, Valrecordl.

Global variables modified: C-row, C-col, CONTENTINDX, V-row, V-col, minindex, page, maxindex, E-over, form-over.

7. PATH-SELECTION;

Function : It positions the cursor in the field value defined by a particular PATH-CODE.

For path-code equal to :

(i) 2, cursor is placed at next available field value

(ii) 3, cursor is placed at same value field of next form

(iii) 4, cursor is placed at next value field of defined sequence.

Global variables accessed : Path-code, maxatt, CONTENTINDX, page, pp, Path-att, Rowdb [Relindx],F-row.

Global variables modified: C-row, C-col, V-row, V-col, path-code, form-over, E-over, F-row, pp.

(B) The SEARCH component is used to find a desired form and display it on the screen. It consists of subroutines FIND-POSITION and FIND-FORM.

1. FIND-POSITION (Var J : integer);

Function: It inputs the keyattributes of a desired tuple, finds the tuple in the W-sheet (by comparing with keyindex [Relindx]) and outputs J integer (equal to Row no. in W-sheet) i.e. the address of the tuple. If tuple is absent then search-fail boolean variable is assigned false.

Global variables accessed : Keyindex [Relindx], Key-col [Relindx), W-maxrows.

Global variable modified : Search-fail.

2. FIND-FORM;

FUNCTION - It finds the desired tuple in W-sheet by FIND-POSITION procedure and display the corresponding form on the screen.

Global variables accessed : Keyindex [Relindx], Key-Col [Relindx], W-maxrows, W-sheet.

Global variables modified : Search-fail, C-row, C-col.

C. The DISPLAY component is used to display a form if the form is small or a part of the form if the form is large. It consists of subroutines, Writechar-form, WRITE-STRING, FIND-MININDEX, DISPLAY as given below.

1. WRITECHAR-FORM (CH:Char);

FUNCTION : It writes CH character at the appropriate position on the screen in accordance with the FIELD FORMAT of blank form by Z[L]. If Z[L] = CHR(O) then the character is written at the extension line.

Global variables accessed : Z[L], L, Val-Create, Form-create, move-out, overlap.

Global variable modified - C-row, C-Col, Z[L].

2. WRITE-STRING (STRING:LINE);

FUNCTION: It writes STRING in the field VAL of a form keeping its format same as field FORMAT of the blank form. The Field FORMAT is made available in the global variable Z.

Global variables accessed : Relindx, W-sheet, Contentindx.

Global variables modified : C-row, C-col.

3. FIND-MININDEX (Var I,J: integer);

FUNCTION : It calculates the first contentindx I and last contentindx J to be displayed on the screen. This is useful to display a large form or hidden part of form during scrolling.

Global variables accessed - FORMS, page, max

4. DISPLAY;

Function : It displays the FORMS datastructure on the terminal screen using WRITE-STRING for displaying FORMS. FORM-CONTENT. VAL.

Global Variables accessed : FORMS, page.

Global Variables modified : Minindex, maxindex, contentindx, C-row, C-col,

D. The ENTRY and EDITING COMPONENT is used to enter or modify a string of characters. It invokes Travelling component implicitely to position the cursor at the desired field of the form, then the movement of cursor within a field is done by MOVE-AHEAD, BACKSPACE subroutines. The DELETECHAR subroutine is used to delete a character while entry and insertion need no separate command and is done by ENTER-INSERT-FORM subroutine. The updating of field format is performed by UPDATE procedure. The various subroutines are given below.

1. MOVE-AHEAD;

   Function : It positions cursor at the next character in the present field. (Here next character is the next of the character pointed by the cursor).
   Global variables accessed : Pointer, length, Form-create, word.
   Global variables modified : C-row, C-col, pointer, over-lap, move-out.

2. BAC [illegible]ACE;

   Function - It POSITIONS CURSOR in the previous character of the present field. (Here the previous character is the previous to the character pointed by the cursor.)
   Global variables accessed : Word, Pointer, L, Z[L], Form-create, Val-update, page.
   Global variables modified : C-row, C-col, L, Pointer.

3. DELETE-CHAR;

   Function - It deletes a character just previous to the present cursor position.
   Global variables accessed : Word, pointer, Val-create, form-create, Val-update, Contentindx, Key-Col[Relindx].
   Global variables modified : Word, pointer,Key-col [Relindx].

4. UPDATE;

   Function - It updates field FORMAT of a blank form specified by CONTENTINDX.

   Global variables that it accesses - Ttyinp, Z[L], L.

   Global variables modified - Z[L],L.

5. ENTER-INSERT-FORM;

   Function - It enters or edits a string of characters 'word' and put the modified string into FORMS, Valrecord and W-sheet.

   Global variables accessed : Word,L,Ttyinp,pointer, Relindx, W-sheet, Val-create, form-create, Val-update.

   Global variables modified : Word,Form-over,E-over, pointer,length,Valrecord,Valrecordl,Forms,W-sheet, Path-code.

(E) The PRINT component is used to get hard-copy of desired no. of forms. It invokes PRINT-FORM subroutine, as given follows :

PRINT-FORM (FILENAME:ALFA,RELNO:integer);

Function : It writes the contents of FORMS in FILENAME.

Global variables it accesses : FORMS, W-Sheet

Global variables modified : Z.

(F) The SAVE and LOAD component allows to save and load the different formating of form in the different files. The saving and loading of data for a relation is also provided by this component. It consists of the subroutines SAVE-FORMAT, SAVE-DATA, LOAD-FORMAT, LOAD-DATA. The description is given below.

1. SAVE-FORMAT (A,I:integer);

   Function : It stores the format of a form (specified by A and I integers in a desired filename).

   Global variables accessed : W-sheet, Maxatt.

   Global variables modified : Temp.

2. SAVE-DATA (A: integer);

   Function : It stores the data of relation 'A' in the Temp.file.

   Global variables accessed : W-sheet, Maxatt.

   Global variable modified : Temp.

3. LOAD-FORMAT;

   Function : It loads the format of a form from the Temp. file.

   Global variables accessed : Temp, Open-set.

   Global variables modified : Maxatt, W-sheet.

4. LOAD-DATA;

   Function : It loads the data of a desired file in the W-sheet.

   Global variables accessed : Temp.

   Global variables modified : W-sheet, Row1, Row2.

As we are simulating the secondary storage in main memory, so some procedures are simulated to provide the data flow from simulated secondary storage to buffer or vice-versa as shown in Fig. (ii). GET-RECORD and PUT-RECORD are used to transfer the data from W-sheet (simulated secondary storage) to Buffer (Adrs) or vice-versa respectively, INSERTRECORD is used to insert a tuple (in the ascending order) in W-sheet. OPEN allows us to open a relation for any legal operation to be done on it. CLOSE procedure is used to close a relation.

In the actual implementation, all these procedures will be modified with the corresponding file access functions. The description of these procedures is given as follows.

1. GET-RECORD (Chno: integer; Var Adrs: Valrec);
   Function: It transfers a tuple located at Rowl of W-sheet (simulated secondary storage) to Adrs. The tuple exists in Chno relation.
   Global variables accessed : W-sheet, Rowl, maxatt.

2. PUT-RECORD (Chno : integer; Adrs: Valrec);
   Function : It transfers a tuple in Adrs to row2 location of W-sheet. The tuple exists in Chno relation.
   Global variables accessed : Row2, maxatt.
   Global variables modified : W-sheet.

For simplicity, some procedures are not represented by COMPONENTS in FUNCTIONAL BLOCK DIAGRAM. These are given as follows.

GET-FORM-CONTENT allows the data to flow from W-sheet to viewing buffer SEND-FORM-CONTENT is used to transfer a form from viewing buffer to Buffer. INIT-FORM is used to fill viewing buffer by a blank form and REDISPLAY-FORM displays this blank form on the screen.

1. GET-FORM-CONTENT;

   Function : It is used to fill field name or field value depending on Row=-1 or other respectively

   Global variables accessed : Maxatt, Relindx, Rowl, W-sheet

   Global variables modified: FORMS.

2. SEND-FORM-CONTENT;

   Function: It sends the set of field name or field value into Valrecordl depending on Rowl=-1 or other respectively.

   Global variables accessed : Rowl, maxatt, FORMS.

   Global variables modified : Valrecordl.

3. INIT-FORM;

   Function : It outputs FORM record for a blank-form

   Global vafiables accessed : W-sheet, Val-create, Form-create, Redis.

   Global variables modified : FORMS, Z, Rowl, contentindx, Valrecord.

3. INSERTRECORD (Chno: integer; Adrs: Valrec);

   Function : It inserts the record of Adrs in W-sheet. The insertion is performed in the ascending order by comparing the keyattributes in keyindex[Chno].

   Global variables accessed : Rowdb[Chno], Key-col[Chno], Keyindex[Chno].

   Global variables modified: Rowdb[Chno], Keyindex[Chno], Row1, Row2.

4. OPEN;

   Function : It opens a relation by adding [Relno] to open-set variable and set the values of maxatt, key-col [Relno] variables.

   Global variables accessed : open-set, W-sheet

   Global variables modified : open-set, maxatt, key-col [Relno].

5. CLOSE;

   Function : It closes the relation by substracting [Relno] from open-set, stores maxatt, key-col[Relno] in W-sheet and reset the values of maxatt and key-col [Relno].

   Global variables accessed : Maxatt, key-col[Relno], open-set,

   Global variables modified : Maxatt, key-col [Relno], W-sheet, open-set.

implemented. Brief summary lines are displayed on the screen. These display the given command and the corresponding dialogue (including the error messages in case of an error). Thus the system is interactive. It provides help at all times. The facilities for printing, saving into a file, loading from a file and redisplaying a desired blank form are also incorporated in the system.

5.2 LIMITATIONS

The following are the limitations in our system.

1. As the present system is screen-oriented, one has to position the cursor on the screen for entering or editing a field of a form. The direct-addressing of the cursor is a hardware feature of a terminal and the command for addressing the cursor may be different for different terminals we used the command for 'DESCOPE-TERMINAL VT52A' in our system, so the system will work only on this terminal.

   In case, one wants to use the system on some other terminal, he should replace the existing command by the actual command in PUTCURSOR procedure of the system.

2. While filling value field of a form, one can't enter first character a '?, because it is used as the help character.

   One may enter '?' by first typing '^X', '^A' etc. and thereafter positioning the cursor at the first character place of the value field.

3. The format of a form can be specified in one fixed sequence only. The sequence is given as follows.

Firstly, one has to specify the Header of the form. Then, he is allowed to type sequence of pairs consisting of field name and value (format). In a pair, firstly the field name thereafter the field value (format) are specified.

This limitation comes due to the existing logic for the creation of form in /C command.

5.3 FUTURE WORKS

The following are the extensions to our system, suggested for the future :

1. The present implementation involves simulation of secondary storage. To make it an actual system, one should use actual secondary storage. The following routines for accessing the secondary storage are needed to be implemented.

A) GETRECORD(Chno, Adrs, Keyval)

B) GETNEXTRECORD(Chno, Adrs)

C) INSERTRECORD (Chno, Adrs)

D) DELETERECORD (Chno, Keyval)

E) UPDATERECORD (Chno, Adrs)

F) MODIFYRECORD (Chno, Adrs)

.) OPENR (Chno, Fname)

H) CLOSER (Chno).

2. In the present implementation, the contents of a form can correspond to the data of a single relation only. A form consisting of data corresponding to multiple relations can be an extension to the present system. In this case, a form will be displayed by fetching the data from various relations. As the modification of contents of a form will involve updating the multiple relations, so modification of contents of a form is difficult.

3. The present system can display only a single form. One may want to view two or more forms at a time. The facility to display more than one forms can be incorporated in the system.

4. Presently, the data is stored in the form of characters in the secondary storage. This may result in more memory requirement for storing the integers of large values. The extension to the present system is to store the integer data in integer form and real data in real form.

5. Presently, printing involves producing the copies of a form, one after the other. If the form is small, one may desire to have some copies at the empty side portion of the page.

Further more extensions are possible in the system depending on the needs of a user and the imagination of a programmer.

REFERENCES

1. Norman Meyrowitz Andries Van Dam, Interactive Editing Systems: Part I and II, acm computing surveys, Volume 14, Number 3, September 1982.

2. Ullman Jefferey, D., Principles of Database Systems, Galgotia Publications, 1984.

3. Marwaha Jugal Kishore, Table Interface and Editor for relational database.